Systèmes de gestion de code source

Matthieu Herrb

CNRS-LAAS

Envol, 22 octobre 2008

`http://www.laas.fr/~matthieu/talks/envol08-sgv.pdf`

Agenda

1. Introduction
2. Concepts d’un système de gestion de version
3. Logiciels pour la gestion de version
   - Modèle local
   - Modèle client-serveur
   - Modèle distribué
4. Conclusion

Agenda

De quoi parle-t-on ?

Système permettant de gérer les modifications d'un ensemble de données.

Typiquement : code source d'un logiciel.

Également applicable à d'autres catégories de données :

- documentation
- site web
- fichiers de configuration d'un système
- etc.

Fonctions de base

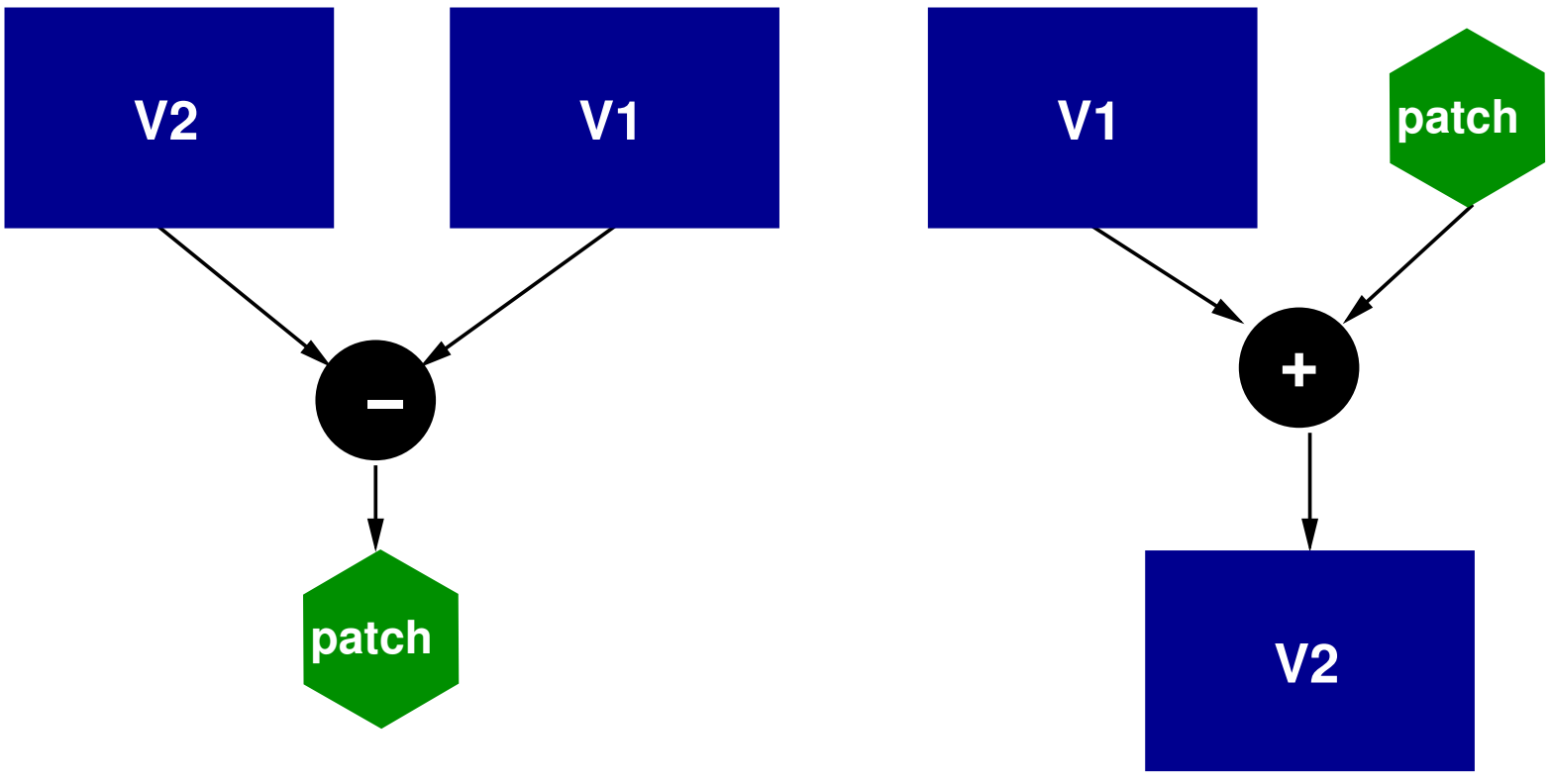
- conserver un historique des modifications
- permettre travailler à plusieurs (verrous, gestion des conflits)
- permettre les modifications en parallèle (branches)
- garantir la sécurité (intégrité, disponibilité, confidentialité)

Comment faire?

- gestion manuelle de plusieurs copies des fichiers
- logiciels dédiés

Agenda

Diff et patch



diff texte

Représentation des différences entre 2 versions d'un fichier.

`diff(1)` produit un diff de 2 fichiers texte :

```
--- a/src/server.c
+++ b/src/server.c
@@ -222,7 +222,9 @@ reset_log(void)
 #ifdef HAVE_SS_LEN
 #define sockaddr_len(s) s.ss_len
 #else
-#define sockaddr_len(s) sizeof(s)
+#define sockaddr_len(s) (s.ss_family == AF_INET6 ? \
+                         sizeof(struct sockaddr_in6) \
+                         : sizeof(struct sockaddr_in))
 #endif

 void
```

commande patch

- *patch :* Pièce qui permet de passer d'une version à la suivante.
- `patch(1)` commande Unix qui utilise un diff comme entrée.

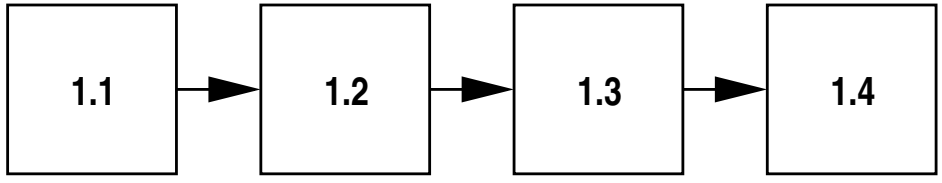
Exemple :

```
# patch -p1 -E < diff
```

`patch` peut gérer des petites incohérences grâce au contexte

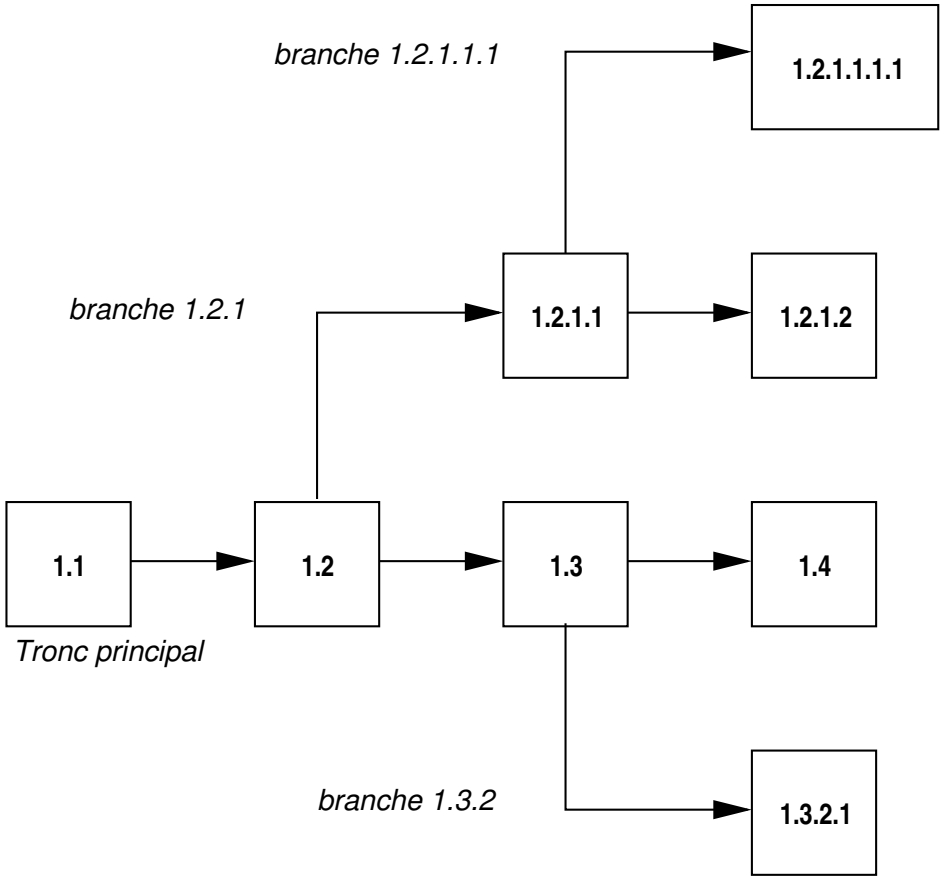
Concepts de base (1)

- **Dépot** (*repository*)
  Répertoire partagé par tous
  Conserve l'historique des modifications
- **Module**
  Ensemble de fichiers sources ou de répertoires constituant un projet.
- **Révision**
  Chaque fichier a un numéro de révision unique.



Concepts de base (2)

- **Branches**



- **Version du projet** $\neq$ révision d'un fichier !

Gestion des branches

Pour :

- corriger un problème sur une ancienne version
- développer 2 idées en parallèle
- gérer sa propre version d'un logiciel
- fusionner après une divergence.

Concepts de base (3)

- **Tags**

  Marques symboliques sur une révision.

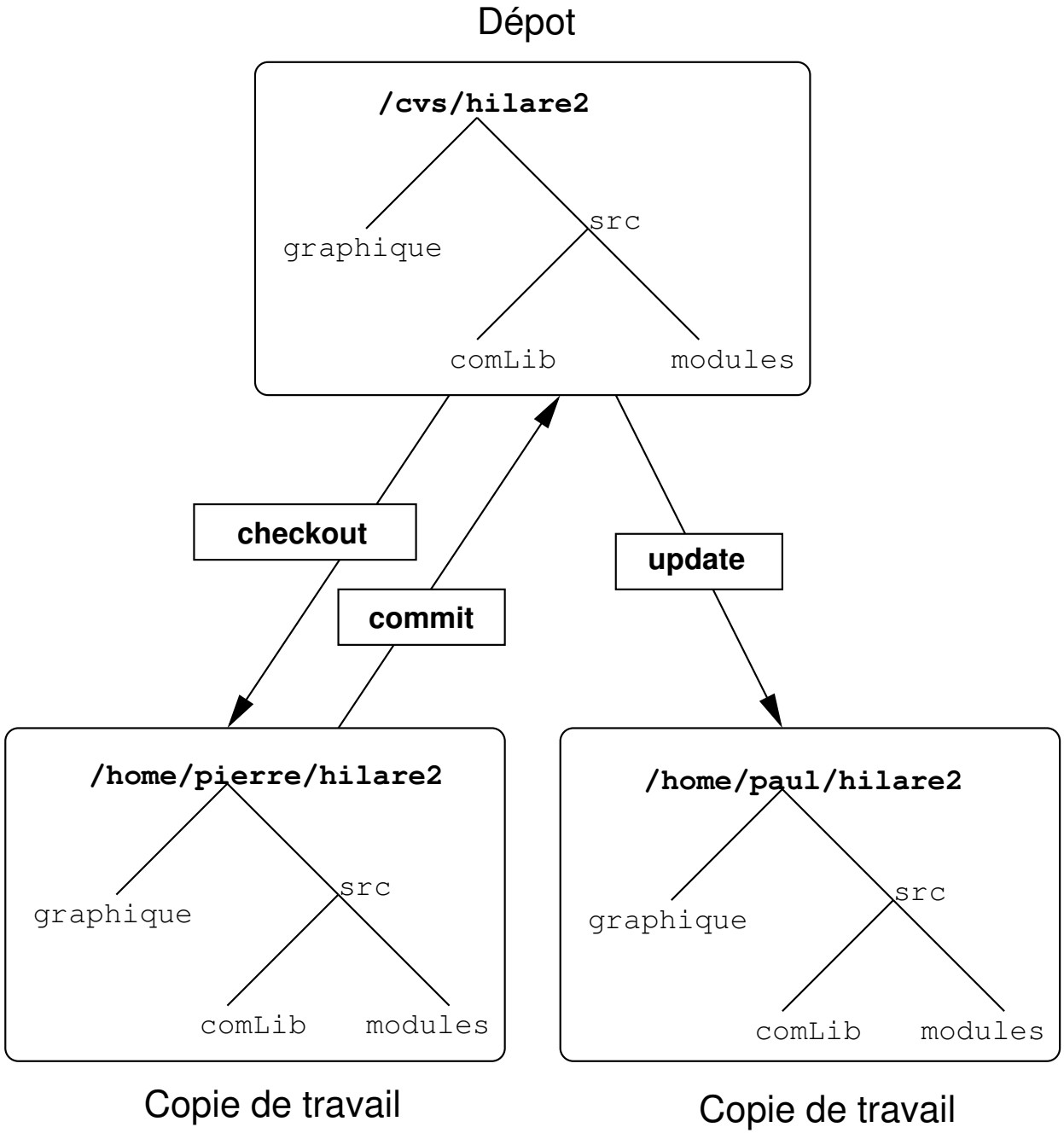
  Permettent de définir les versions du projet.

  Permettent de nommer des branches.

Travailler à plusieurs

- Pas de verrou sur les sources.
  Chacun a sa propre copie.
- Gestion des conflits:
  - pas de `commit` possible sans `update`
  - fusion automatique
  - détection des conflits $\longrightarrow$ résolution à la main
  - pas de nouveau `commit` avant résolution du conflit

Exemple


Dépot
/cvs/hilare2
graphique
src
comLib
modules
checkout
commit
update
/home/pierre/hilare2
graphique
src
comLib
modules
/home/paul/hilare2
graphique
src
comLib
modules
Copie de travail
Copie de travail

Autres fonctions d'un SGV

- Visualisation de l'historique sous diverses formes
- Exécution automatique de scripts avant/après commit
  - Tests de validation,
  - Envoi d'e-mail après commit.
- Annotation du code avec les contributions
- Recherche dichotomique de regressions
- Import/export vers d'autres SGV
- ...

Agenda

Trois modèles de fonctionnement

Local

Fonctionne dans un système de fichiers local. Pas de réseau.

- SCCS, RCS,...

Client/Serveur (ou centralisé)

Un serveur centralise le dépot, accessible à distance.

- CVS
- Subversion

Distribué

Multiples copies du dépots, branches locales.

- bitkeeper, monotone, arch, darcs
- mercurial, git, bazaar

Critères de choix

- Modèle de développement (Centralisé ou non)
- Souplesse d'utilisation :
  - gestion des branches (fusion)
  - déplacement/renommage des fichiers
- Sécurité
  - intégrité : signature des fichiers, des commits
  - gestion explicite des droits d'accès par utilisateur
  - disponibilité : possibilité de récupération des données en cas de corruption du dépot
- Efficacité, Vitesse - important pour des gros projets
- Diffusion, développement actif
- Portabilité (Mac OS X/Unix/Windows).

SCCS, RCS

SCCS

SGV historique d'Unix.
défini les concepts de base de beaucoup de SGVs.
Longtemps pas Open Source
Verrouillage très strict $\longrightarrow$ pas de fusion.

GNU RCS

Extension de SCCS. Introduit la notion de fusion.
Gestion des branches très lourde.

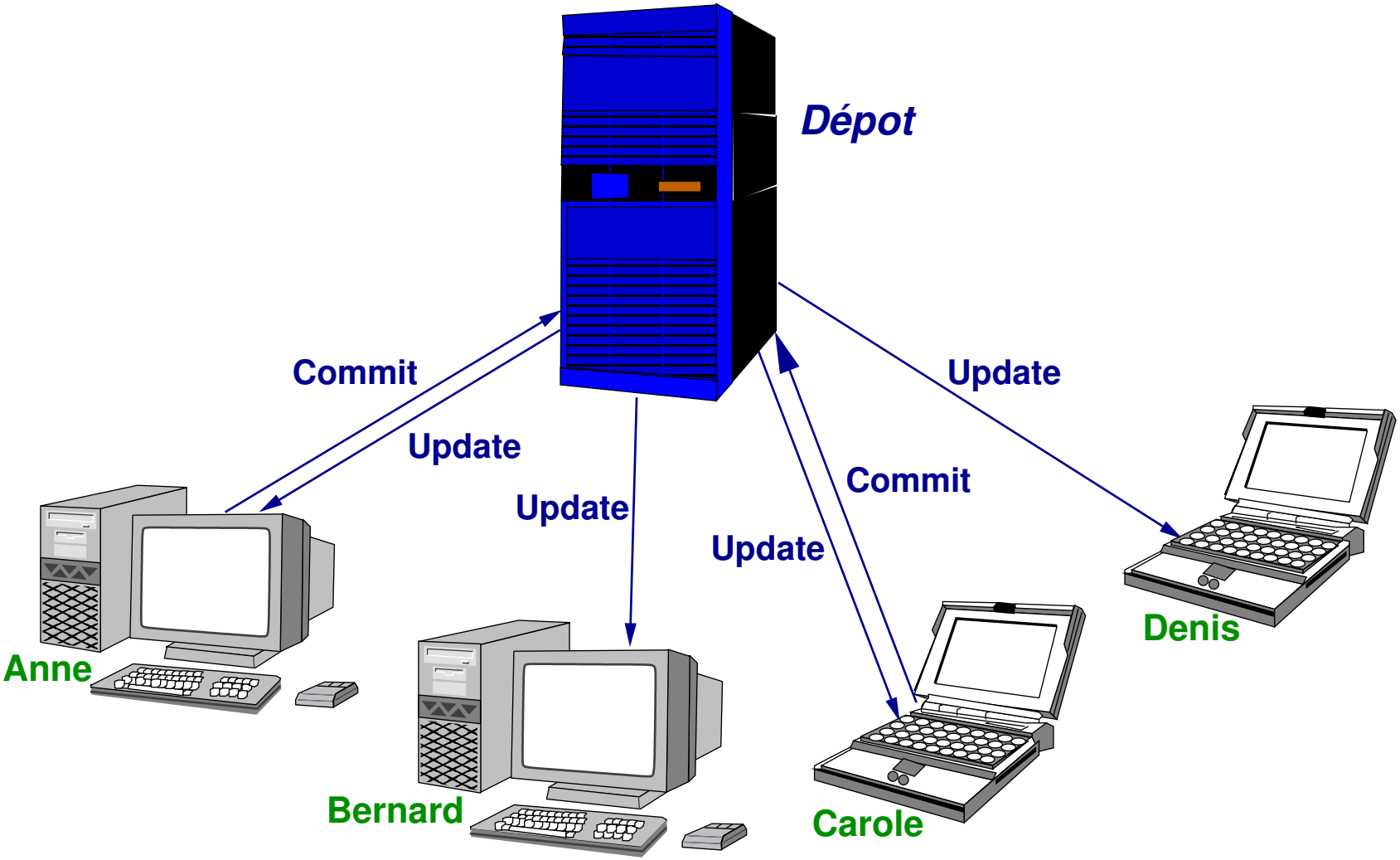
Principales limites : un seul utilisateur à la fois, une seule copie de travail.

Reste intéressant pour certains cas simples.

Client-serveur - principe

- Dépot stocké dans un endroit partagé
  - par le système de fichiers
  - par un mécanisme réseau (rsh/ssh ou protocole dédié)
- Plusieurs copies de travail en parallèle : opérations de fusion.
- Nécessaire d'avoir la connexion au dépot pour committer.
- Le «tronc» a une importance particulière : modèle très centralisé.

Client-serveur - principe



CVS

Concurrent **V**ersion **S**ystem
`http://www.nongnu.org/cvs/`

- Basé sur RCS. Centralise le dépot et autorise plusieurs copies de travail concurrentes.
- Initialement uniquement local dans un système de fichiers.
- Mode client/serveur simple
- Ne gère pas l'authentification - nécessite un compte Unix par utilisateur sur le serveur
- Travaille fichier par fichier. Pas de commits atomiques, ne gère pas les rennomages ou les déplacements de fichiers
- Notion de "vendor branch"
- Gestion des branches très lourde. Pas adaptée pour des développements parallèles.

OpenCVS

Réimplémentation de GNU CVS sous licence BSD.

- Compatible au niveau dépot, commandes, protocole.
- Meilleur contrôle de la sécurité.
- Extensions prévues : commits atomiques, support du renommage.

Subversion

`svn`
`http://subversion.tigris.org/`

- «Successeur» de CVS
- Interface utilisateur similaire à CVS
- Commits atomiques
- Gère le renommage de fichiers
- Gère les meta-données (droits d'accès, propriétaire)
- Meilleure gestion des branches, mais pas de mémoire des fusions
- Stockage sophistiqué (base de données,...)
- Accès distants via HTTP/DAV

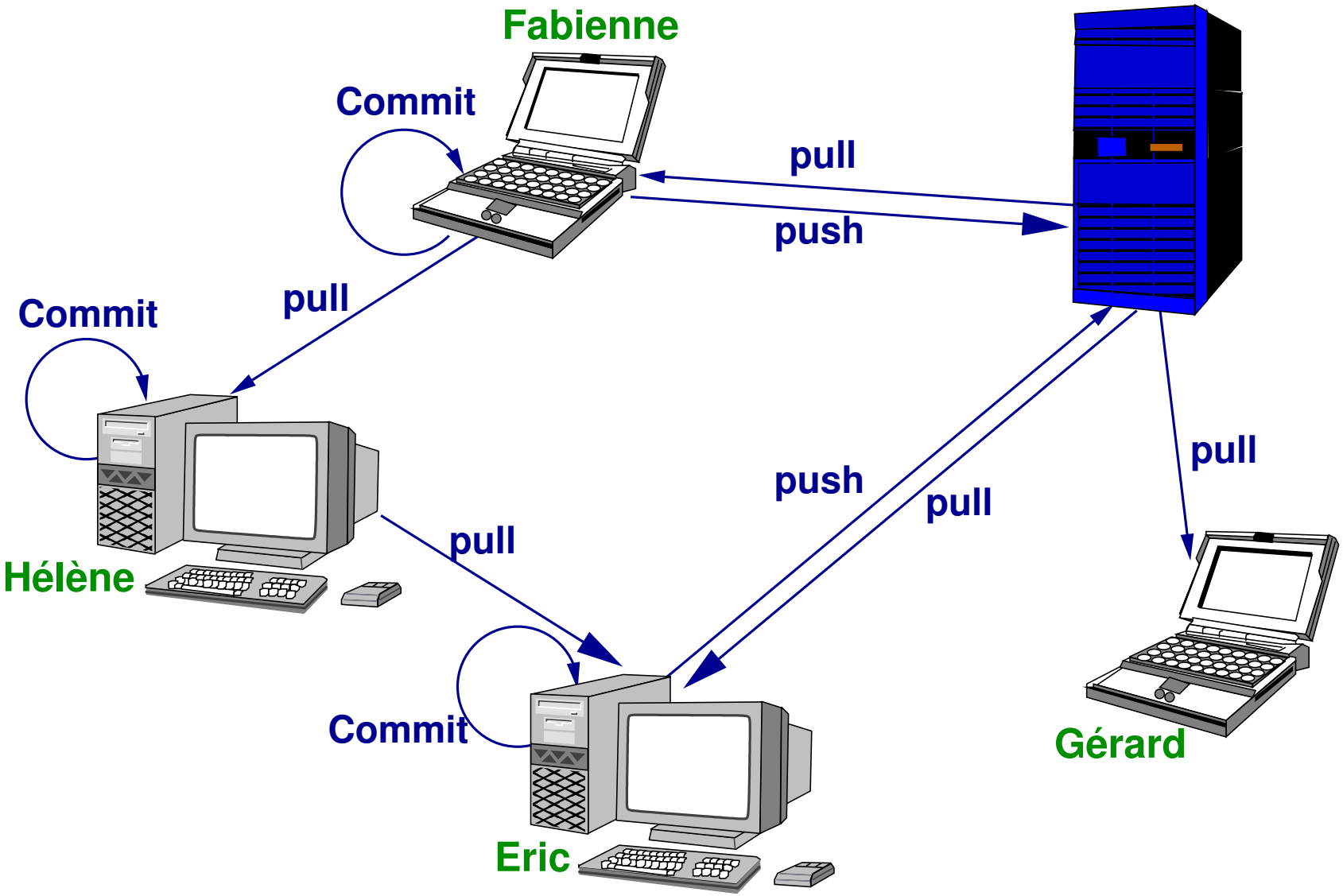
Perforce

- Propriétaire
- Licences gratuites pour projets Open Source (FreeBSD, Perl, par exemple).
- Rapide
- Branches faciles et peu coûteuses
- Bon algorithme de fusion

Systèmes distribués - principe

- Plus de dépot centralisé
- Chaque développeur a sa copie avec ses branches privées
- Opérations push/pull : synchronisation avec les autres dépots.
- Simplification de la fusion de branches en gardant l'historique des fusions.
- Influence sur la philosophie de développement : plus de liberté,
  mais risque de dispersion...

Systèmes distribués - principe



Le premier SGV distribué : BitKeeper

- Développé par Larry McVoy (Sun, Sgi,...) pour le noyau Linux.
- Basé sur SCCS en ajoutant la notion de réplication du dépot et le suivi des méta-données.
- Le premier SGV à utiliser une copie du dépot par branche.
- Produit non libre avec une licence très restrictive.
- Abandonné par Linux après le fiasco.

git

`http://git.or.cz/`

Développé par Linus Torvalds pour remplacer BitKeeper.

- Architecture à deux couches :
  - la plomberie : un système de fichiers adressable par le contenu (via hashs SHA-1) avec gestion de l'historique.
  - la porcelaine : les commandes de haut-niveau destinées à l'utilisateur normal de git. très proche de Mercurial.
- Utilisation systématique de crypto. Un objet qui rentre dans git est immuable.
- Très rapide.
- Utilisé par de nombreux projets : Linux, X.Org, Mesa3d, ...
- Développement très actif.

Mercurial

`hg`
`http://www.selenic.com/mercurial/`

- Écrit en python
- Utilise des hash SHA-1
- Relativement compact
- Extensible (framework pour des extensions)
- Développement actif
- Utilisateurs : OpenSolaris, Xen
- Projet encore jeune.
- Gestion des renommages par copie.

Baazar

`bzr`

`http://www.bazaar-vcs.org/`

- Réécrit en python
- Sponsorisé par Cannonical (Ubuntu)
- Branches externes (utilisant d'autres SGV) (bientôt...)
- Utilisateurs : Ubuntu launchpad, Drupal, Samba

Agenda

Conclusion

- Technologie en pleine (r)évolution.
- L'apparition depuis 2002 des SGV distribués change la façon de travailler.
- Attention à la pérénité.

Questions ?

Le fiasco de BitKeeper

- Linux Torvalds a commencé à utiliser BK vers 2002 (auparavant il n'utilisait pas de SGV pour Linux!)
- De nombreux mécontents (à cause de la licence), encourageant le développement d'autres SGV distribués (Arch, Darcs, Mercurial)
- En 2005, Andrew Tridgell (Samba) commence le développement d'un client libre compatible BK. Larry McMoy révoque toutes les licences gratuites de BK. Linux n'a à nouveau plus de SGV.
- Linus teste les SGV distribués existants. Aucun ne convient à ses besoins (soit trop limités, soit trop lents, soit trop complexes)

$\longrightarrow$ Linus commence alors le développement de son propre SGV distribué : **git**.